Parte a.- Una función booleana (tipo *BoolFun*) recibe como parámetro un arreglo *x* de *n* variables booleanas (es decir cada variable puede ser solo verdadero o falso) y entrega un resultado booleano calculado a partir de las variables en *x* operadas con `&&`, `||` y `!`. Por ejemplo *f4* es una función booleana que recibe un arreglo de 4 variables:

```
int f4(int x[]) {
  return (x[0]||!x[1]) &&
         (!x[1]||x[2]||!x[3]);
}
```

La función *recuento* de más abajo evalúa una función booleana *f(x)* para cada una de las 2^n combinaciones posibles de valores que pueden tomar las *n* variables del arreglo *x*, retornando el número de veces en que la función es verdadera. Esta función es lenta de calcular ya que toma tiempo $O(2^n)$.

```
typedef int (*BoolFun)(int x[]);
int cnt= 0;
void gen(int x[], int i, int n,
         BoolFun f) {
  if (i==n) {
    if ((*f)(x))
      cnt++;
  }
  else {
    x[i]= 0; gen(x, i+1, n, f);
    x[i]= 1; gen(x, i+1, n, f);
} }
int recuento(int n, BoolFun f) {
  int x[n];
  gen(x, 0, n, f);
  return cnt;
}
```

Paralelice la función *recuento* considerando una máquina octa-core. Es decir contabilice el número de veces en que *f* se hace verdadera usando 8 threads.

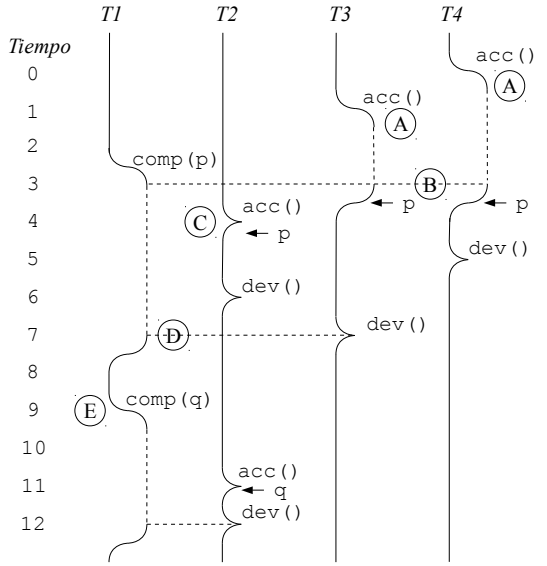
Ayuda: Programe una función recursiva que genere las 8 combinaciones posibles de valores de un arreglo *z* de 3 variables booleanas (de manera similar a como lo hace la función *gen* de más arriba). Para cada una de estas 8 combinaciones (i) cree un arreglo *x* de *n* variables, en donde las primeras 3 variables están fijas con los valores de *z*, y (ii) cree un thread que invoque otra función recursiva que genere las 2^{n-3} combinaciones posibles de valores de las variables *x[3]*, …, *x[n-1]*. Para cada una de estas 2^{n-3} combinaciones invoque *f(x)* contabilizando los casos en que la función es verdadera.

Parte b.- Programe las siguientes funciones cuyo fin es permitir que varios threads compartan datos en modo lectura:

- *void compartir(void *ptr)*: Ofrece compartir los datos apuntados por *ptr* con los threads que llamen a *acceder*. *Compartir* queda en espera hasta que los threads notifiquen que desocuparon los datos llamando a *devolver*.
- *void *acceder( )*: Solicita acceso a los datos ofrecidos con *compartir*. Si hay una llamada a *compartir* en espera, retorna de inmediato el puntero *ptr* suministrado mediante *compartir*. Si no, espera hasta la próxima invocación de *compartir*.

void devolver( ): Notifica que los datos compartidos ya no se usarán.

El diagrama de arriba a la derecha explica el funcionamiento pedido. En A *acceder* se bloquea hasta que otro thread invoque *compartir*. Esto ocurre en B, lo que hace que todos los threads que esperaban en una llamada a *acceder* se desbloqueen retornando el puntero a los datos (p en este caso). La llamada a *compartir* queda en espera hasta que todos los threads que llamaron a *acceder* notifiquen que no usarán más los datos invocando *devolver*. En C, como hay una llamada a *compartir* en espera, *acceder* retorna de inmediato los datos compartidos. En D se invoca el último *devolver*, y por lo tanto *compartir* retorna. Si se invoca *compartir* y no hay threads que llamaron a acceder, *compartir* espera hasta que algún thread invoque acceder (ver E).



Para la sincronización use un *mutex* y una condición de *pthreads*, ambos almacenados en variables globales.

Su tarea debe compilar sin warnings en anakena, pero no necesita correr exitosamente en ella porque esta máquina no permite crear demasiados threads.

Recursos

Baje *t3.zip* de material docente en U-cursos y descomprímalo. El directorio contiene los archivos *test-t3.c*, *Makefile* y *t3.h*. Ud. debe programar las funciones pedidas en el archivo *t3.c*. Use *Makefile* para compilar y ejecute su tarea invocando *./test-t3*. Se le felicitará si su tarea funciona de acuerdo al enunciado.

Entrega

Ud. debe entregar el archivo *t3.c* por medio de U-cursos. No incluya otros archivos por favor. No se aceptarán tareas que no pasen los tests en el archivo *test-t3.c*. Se descontará medio punto por día de atraso. No se consideran los días sábado, domingo o festivos.